

# Optimal Symbol Alignment Distance: A New Distance for Sequences of Symbols

Javier Herranz, Jordi Nin, Marc Solé

## Abstract

Comparison functions for sequences (of symbols) are important components of many applications, for example clustering, data cleansing and integration. For years, many efforts have been made to improve the performance of such comparison functions. Improvements have been done either at the cost of reducing the accuracy of the comparison, or by compromising certain basic characteristics of the functions, such as the triangular inequality.

In this paper, we propose a new distance for sequences of symbols (or strings) called Optimal Symbol Alignment distance (OSA distance, for short). This distance has a very low cost in practice, which makes it a suitable candidate for computing distances in applications with large amounts of (very long) sequences. After providing a mathematical proof that the OSA distance is a real distance, we present some experiments for different scenarios (DNA sequences, record linkage, ...), showing that the proposed distance outperforms, in terms of execution time and/or accuracy, other well-known comparison functions such as the Edit or Jaro-Winkler distances.

## Index Terms

Sequences of Symbols, String Distances, Triangular Inequality

## I. INTRODUCTION

Sequences of symbols are a well-known data representation type and are widely used in databases for representing many types of non numerical attributes, such as names or addresses.

J. Herranz is with the Dept. Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, in Barcelona (Spain)

J. Nin is with CNRS; LAAS; 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4 (France).

M. Solé is with the Dept. Arquitectura de Computadors, Universitat Politècnica de Catalunya, in Barcelona (Spain)

E-mail addresses: jherranz@ma4.upc.edu, jnin@laas.fr, msol@ac.upc.edu

Moreover, they are gaining more and more attention in many other communities because they can represent data in a large variety of domains, such as gene information [1], vehicular tracking [2] or sequential patterns [3].

For this reason, there is a lot of work in computing similarities among sequences of symbols [4], [5], [6], [7], [8], [9], [10], [11]. However, the similarity measures presented in most of those works either do not fulfill the mandatory conditions to be a real distance (most of the times, because the triangular inequality does not hold) or do not contain a proof for that. Formally, a distance function  $d$  must satisfy the following properties:

- 1) Symmetry:  $d(A, B) = d(B, A)$  for all sequences  $A, B$
- 2) Positivity:  $d(A, B) \geq 0$  for all sequences  $A, B$
- 3) Reflexivity:  $d(A, A) = 0$  for all sequence  $A$
- 4) Triangular Inequality:  $d(A, B) \leq d(A, C) + d(B, C)$  for all sequences  $A, B, C$

To the best of our knowledge, there are only two sequence measures that fulfill these conditions: the Hamming distance [6] and the Levenshtein (Edit) distance [8]. The remaining measures are similarity functions instead of real distances because they do not comply with the triangular inequality (or this is not proved). For this reason the application of such measures to the scenarios where having a metric space is a must, such as metric spaces [12], clustering [13] or k-nearest neighbors algorithms [14], becomes unfeasible from a theoretical point of view.

The Hamming and Edit distances present also some problems. For instance, the Hamming distance can only be applied to sequences of the same length, while the Edit distance has a large, both practical and theoretical, complexity ( $O(n^2)$ ). For these reasons many similarity measures have been developed, albeit sacrificing some of the mandatory properties of a distance. For example, the Jaro-Winkler distance [7] is very efficient in terms of practical computational cost when the compared strings are not too large. Therefore, it saves execution time (when compared to Edit distance) in applications where there are many comparisons to be done.

In this paper, we present a new string comparison function with a very low practical cost (as the Jaro-Winkler distance). This new distance, that we call *Optimal Symbol Alignment* distance (OSA, for short), is a real distance (as the Edit distance), because it fulfills all the properties defined before. In some sense, the OSA distance enjoys the best of the two worlds: it can be used in scenarios where triangularity is a must, and it also saves execution time in applications with a large number of comparisons. For this reason, we believe that the new distance may be

of great interest in a wide range of practical situations. We will describe some experiments to show the applicability of the OSA distance to different scenarios: computing distances between long DNA sequences, and performing record linkage in both medium and large databases. The experiments show that the OSA distance always outperforms the Edit and Jaro-Winkler distances in terms of execution time, sometimes by several orders of magnitude. Moreover, in terms of quality, the results obtained with the OSA distance are very similar (if not better) than those obtained with the Edit distance, except for a specific (hard) scenario of record linkage without filtering.

The rest of this paper is organized as follows. Firstly, in Section II we introduce some basic concepts about the Edit and Jaro-Winkler distances. Then, in Section III we provide the definition of the OSA distance, as well as a mathematical proof that it satisfies the triangular inequality. Implementation details for an efficient computation of the OSA distance are given in Section IV, including two new algorithms for finding the optimal alignment of a symbol (in our particular scenario). Next, in Section V we explain some experiments that we have run to show the applicability of the OSA distance and to compare it with the Edit and Jaro-Winkler distances. Finally, we present some conclusions and possible lines of future work in Section VI.

## II. PRELIMINARIES

In this section, we recall some basic notions about two classical distances: the Edit and Jaro-Winkler ones.

### A. Edit Distance

The Edit distance [8], [11], [15] measures the difference between two sequences, given by the minimum number of edit operations needed to transform one sequence into the other. An edit operation can be either an insertion, deletion or substitution of a single symbol, although many variations exist in which the set of allowed operations is larger or more restricted. In some way, Edit distance assumes that the differences between two sequences are due to typos or spelling errors.

The Edit distance has found a large variety of applications in many scenarios and has achieved very good results [16]. However, the Edit distance has a large complexity: its computation using classical algorithms [17] based on dynamic programming has a complexity equal to  $O(n^2)$ ,

where  $n$  is the size of the shortest string. Other algorithms to compute the Edit distance exist [18], [19], having a lower complexity of  $O(dn)$ , for example, where  $d$  is the real Edit distance. Note that, when two completely different strings have to be compared, the complexity of these variants is the same as that of the classical algorithm.

### B. Jaro-Winkler Distance

Due to the high computational cost of the Edit distance, other sequence comparison functions are preferred in some situations requiring intensive distance computations. An example is the Jaro-Winkler distance [7]. The computation of this distance comprises three basic steps: (i) compute the string lengths, (ii) find the common characters in the two strings, and (iii) count the number of transpositions. A common character is a character placed closely in both strings, where closely means that the difference between both positions is less than half of the shortest string length. A transposition occurs when a common character in one string is out of order with respect to the corresponding character in the other string. Although often referred as a distance, the Jaro-Winkler is actually not a distance in the mathematical sense of the term, because it does not fulfill the triangular inequality. Therefore, it cannot be applied to any problem where triangularity is a must, for example in clustering or metric spaces. However, since the Jaro-Winkler distance is very efficient in terms of computational cost, it is preferred to the Edit distance in some scenarios.

## III. OPTIMAL SYMBOL ALIGNMENT (OSA) DISTANCE

The intuition behind the new Optimal Symbol Alignment (OSA) distance is that strings are close if they have many common symbols, and in addition their common symbols are placed in similar positions, in the strings being compared.

Given a finite alphabet of symbols  $X$ , let  $A = (a_1, \dots, a_{n_A})$  and  $B = (b_1, \dots, b_{n_B})$  be two sequences of symbols, where  $a_i, b_j \in X$ , for  $i = 1, \dots, n_A$ ,  $j = 1, \dots, n_B$ . For any sequence of symbols  $A$ , we define as  $X_A \subseteq X$  the subset of symbols that appear in  $A$ ; that is,  $X_A = \{x \in X \text{ s.t. } \exists i \in \{1, 2, \dots, n_A\} \text{ with } a_i = x\}$ . For a symbol  $x \in X_A$ , we also define the subset of positions  $A_x = \{i \in \{1, \dots, n_A\} \text{ s.t. } a_i = x\}$ .

We define the OSA distance  $d(A, B)$  between the sequences  $A$  and  $B$  as

$$d(A, B) = \sum_{x \in X_A \cup X_B} d(x, A, B),$$

where the value  $d(x, A, B)$  is defined as

$$d(x, A, B) = \begin{cases} |A_x| & \text{if } x \in X_A - X_B \\ |B_x| & \text{if } x \in X_B - X_A \\ f(x, A, B) & \text{if } x \in X_A \cap X_B \end{cases}$$

Finally, we have to define the value of  $f(x, A, B)$ , which is the contribution of the symbol  $x$  to the distance  $d(A, B)$ , when this symbol  $x$  is included in both sequences  $A$  and  $B$ . Let us assume without loss of generality that  $|A_x| \leq |B_x|$ . The idea is to select the subset of  $|A_x|$  positions  $j$ , from the set  $B_x$ , which are globally closest to the set of  $|A_x|$  positions in  $A_x$ . Namely, if  $i_1 < i_2 < \dots < i_{|A_x|}$  are the positions in  $A_x$ , then we select  $|A_x|$  positions  $j_1 < j_2 < \dots < j_{|A_x|}$  in  $B_x$  minimizing the global distance  $|i_1 - j_1| + \dots + |i_{|A_x|} - j_{|A_x|}|$ . We use notation  $j_h = \text{pj}(i_h, A, B)$ , for  $h = 1, \dots, |A_x|$ , to denote the position in  $B_x$  that optimally matches position  $i_h \in A_x$ . We say that  $j_h$  is the *projection* of position  $i_h$  from sequence  $A$  to sequence  $B$ . For completeness, we also use the symmetric notation  $i_h = \text{pj}(j_h, B, A)$ .

Each of these common symbols  $a_{i_h} = b_{j_h} = x$ , for  $h = 1, \dots, |A_x|$ , will contribute with  $\frac{|i_h - j_h|}{n_{AB}}$  to the value  $f(x, A, B)$ , where  $n_{AB} = \max\{n_A, n_B\}$ . In this way, we ensure that these contributions are bounded by 1. The remaining  $|B_x| - |A_x|$  symbols will be considered as non-common symbols, so each of them will contribute with a 1 to the global distance  $d(A, B)$ .

Taking all these facts into account, we finally have

$$f(x, A, B) = (|B_x| - |A_x|) + \frac{1}{n_{AB}} \sum_{i_h \in A_x} |i_h - \text{pj}(i_h, A, B)|.$$

Depending on the differences between the two sequences to be compared (more or less repeated symbols, more or less transpositions, etc.) the OSA distance  $d_{\text{OSA}}(A, B)$  will be more or less similar to the Edit distance  $d_{\text{Edit}}(A, B)$ . But in any case, they will not be very far, because it is easy to prove that  $\frac{d_{\text{Edit}}(A, B)}{2} \leq d_{\text{OSA}}(A, B) \leq 2 \cdot d_{\text{Edit}}(A, B)$ , for any two sequences  $A, B$ .

### A. Proving the Triangular Inequality

It is straightforward to check that the function  $d(A, B)$  defined in the previous section satisfies the properties of symmetry, positivity and reflexivity. Let us show that it also satisfies the triangular inequality property. Let  $A, B, C$  be three arbitrary sequences of symbols:  $A = (a_1, \dots, a_{n_A})$ ,  $B = (b_1, \dots, b_{n_B})$  and  $C = (c_1, \dots, c_{n_C})$ . We want to prove that  $d(A, B) \leq d(A, C) + d(B, C)$ .

Let us first consider the particular case where each symbol  $x \in X$  can appear at most once in each sequence (i.e. there are no repetitions of symbols in any single sequence).

*Proposition 1:* If  $|A_x| \leq 1$ ,  $|B_x| \leq 1$  and  $|C_x| \leq 1$ , for every symbol  $x \in X$ , then  $d(A, B) \leq d(A, C) + d(B, C)$ .

*Proof:*

For each symbol  $x \in X_A \cup X_B$ , we have one of the three following cases:

- 1)  $x \in X_A - X_B$ , then the contribution of  $x$  to  $d(A, B)$  is exactly 1. We have either  $x \in X_C$ , which implies the contribution of  $x$  to  $d(B, C)$  is exactly 1, or  $x \notin X_C$ , which implies the contribution of  $x$  to  $d(A, C)$  is exactly 1. In both cases, the contribution of  $x$  to  $d(A, C) + d(B, C)$  is greater or equal than the contribution of  $x$  to  $d(A, B)$ .
- 2)  $x \in X_B - X_A$ , symmetric case.
- 3)  $x \in X_A \cap X_B$ . In this case, we have that the contribution of  $x$  to  $d(A, B)$  is  $d(x, A, B) = \frac{|i-j|}{n_{AB}} \leq 1$ , where  $A_x = \{i\}$  and  $B_x = \{j\}$ . If  $x \notin X_C$ , then the contribution of  $x$  to  $d(A, C) + d(B, C)$  is 2. If  $x \in X_C$ , say  $C_x = \{k\}$ , we have  $x \in X_A \cap X_B \cap X_C$ , and we can write

$$f(x, A, B) = \frac{|i-j|}{n_{AB}} = \frac{|i-k+k-j|}{n_{AB}} \leq \frac{|i-k|}{n_{AB}} + \frac{|k-j|}{n_{AB}}.$$

Now we can consider two different cases. The first one is when  $n_C \leq n_{AB}$ . In this case, we have  $n_{AC} \leq n_{AB}$  and  $n_{BC} \leq n_{AB}$ , and so the above value  $f(x, A, B)$  is less than or equal to

$$\leq \frac{|i-k|}{n_{AC}} + \frac{|k-j|}{n_{BC}} = f(x, A, C) + f(x, B, C).$$

Now for the second case, where  $n_C = n_{AB} + \ell$ , for some integer  $\ell > 0$ , there are  $\ell$  symbols in  $X_C$  that are not in  $X_A \cup X_B$ . Now we have  $n_{AC} = n_{BC} = n_{AB} + \ell$ . Note that these  $\ell$  symbols will not contribute to the value  $d(A, B)$ , but will contribute with  $2\ell$  to the value  $d(A, C) + d(B, C)$ .

Let us go back to our situation where  $x \in X_A \cap X_B \cap X_C$ , we have

$$f(x, A, B) \leq \frac{|i-k|}{n_{AB}} + \frac{|k-j|}{n_{AB}} = \frac{|i-k|}{n_{AC} - \ell} + \frac{|k-j|}{n_{BC} - \ell}.$$

Now we can use the fact that  $\frac{a}{b-\ell} = \frac{a}{b} + \frac{a\ell}{b(b-\ell)}$  and so the last inequality becomes

$$f(x, A, B) \leq \frac{|i-k|}{n_{AC}} + \frac{|i-k| \cdot \ell}{n_{AC}(n_{AC} - \ell)} + \frac{|k-j|}{n_{BC}} + \frac{|k-j| \cdot \ell}{n_{BC}(n_{BC} - \ell)} \leq$$

$$\begin{aligned} &\leq \frac{|i-k|}{n_{AC}} + \frac{|k-j|}{n_{BC}} + \frac{\ell}{n_{AC}-\ell} + \frac{\ell}{n_{BC}-\ell} \\ &= f(x, A, C) + f(x, B, C) + \frac{\ell}{n_{AC}-\ell} + \frac{\ell}{n_{BC}-\ell}. \end{aligned}$$

Here we have used that  $|i-k| \leq n_{AC}$  and  $|k-j| \leq n_{BC}$ .

If we consider all the symbols  $x \in X_A \cap X_B \cap X_C$ , together, we have

$$\begin{aligned} \sum_{x \in X_A \cap X_B \cap X_C} f(x, A, B) &\leq \left( \sum_{x \in X_A \cap X_B \cap X_C} f(x, A, C) + \sum_{x \in X_A \cap X_B \cap X_C} f(x, B, C) \right) + \\ &+ |X_A \cap X_B \cap X_C| \cdot \left( \frac{\ell}{n_{AC}-\ell} + \frac{\ell}{n_{BC}-\ell} \right) \end{aligned}$$

But we can now use the fact that  $\ell + |X_A \cap X_B \cap X_C| \leq n_C = n_{AC} = n_{BC}$ , which implies that the last part of the expression above is less than or equal to  $2\ell$ . Recall that the  $\ell$  symbols which are in  $X_C - (X_A \cup X_B)$  contribute with  $2\ell$  to the value  $d(A, C) + d(B, C)$ . Summing up, if we consider the symbols  $x \in X_A \cap X_B \cap X_C$ , their contribution to  $d(A, B)$  is less than or equal to the contribution of these symbols to  $d(A, C) + d(B, C)$  plus the contribution of the symbols in  $X_C - (X_A \cup X_B)$  to  $d(A, C) + d(B, C)$ .

Putting all the pieces together, we finally have that  $d(A, B) \leq d(A, C) + d(B, C)$  always holds, as desired. ■

Now we can prove that the triangular inequality holds for any triple of sequences, even if they have repeated symbols.

*Theorem 1:* Let  $A, B, C$  be three arbitrary sequences of symbols in a finite alphabet  $X$ . Then  $d(A, B) \leq d(A, C) + d(B, C)$ .

*Proof:* The idea is to construct, from the initial sequences  $A = (a_1, \dots, a_{n_A})$ ,  $B = (b_1, \dots, b_{n_B})$  and  $C = (c_1, \dots, c_{n_C})$ , new sequences  $A' = (a'_1, \dots, a'_{n_A})$ ,  $B' = (b'_1, \dots, b'_{n_B})$  and  $C' = (c'_1, \dots, c'_{n_C})$  such that  $A', B', C'$  do not have repeated symbols, and then to apply Proposition 1 to  $A', B', C'$ .

Specifically, for each symbol  $x \in X_C$ , let  $c_{k_1}, \dots, c_{k_{|C_x|}}$  be the list of  $|C_x|$  letters in  $C$  which are equal to  $x$ , such that  $k_1 < k_2 < \dots < k_{|C_x|}$ . We replace  $c_{k_h} = x$  with  $c'_{k_h} = x|h$ , for  $h = 1, \dots, |C_x|$ . Now, if  $x \in X_A$ , we consider  $i_h = \text{pj}(k_h, C, A) \in A_x$ , for  $h = 1, \dots, |C_x|$ ,

and we replace  $a_{i_h} = x$  with  $a'_{i_h} = x|h$ , as well. Note that, when  $|C_x| > |A_x|$ , there are some positions  $h \in \{1, \dots, |C_x|\}$  for which  $i_h$  is not defined. On the other hand, when  $|C_x| < |A_x|$ , there are some  $x$  symbols in  $A$  that have not been modified. If this is the case, we replace these symbols  $a_{i_h} = x$  with pairwise different symbols  $a'_{i_h} = x|h$ , for  $h = |C_x| + 1, \dots, |A_x|$ .

Exactly the same process is applied to the symbols in  $B$  which are equal to  $x$ . Finally, if there is a symbol  $x \in X_A$  such that  $x \notin X_C$ , then we replace  $a_{i_h} = x$  with  $a'_{i_h} = x|h$ , for  $h = 1, \dots, |A_x|$ . The same is done for symbols  $x \in X_B - X_C$ .

At the end of this process, we have new sequences  $A' = (a'_1, \dots, a'_{n_A})$ ,  $B' = (b'_1, \dots, b'_{n_B})$  and  $C' = (c'_1, \dots, c'_{n_C})$  such that none of them have repeated symbols. We can therefore apply Proposition 1, to deduce  $d(A', B') \leq d(A', C') + d(B', C')$ .

Because of the way in which we have constructed  $A', B', C'$ , we have  $d(A', C') = d(A, C)$  and  $d(B', C') = d(B, C)$ . Furthermore, we obviously have  $d(A, B) \leq d(A', B')$ , because maybe the new names assigned to letters in  $A$  and  $B$  do not correspond with the optimal matching between these two sequences.

Summing up, we have

$$d(A, B) \leq d(A', B') \leq d(A', C') + d(B', C') = d(A, C) + d(B, C),$$

as desired. ■

*1) A Simple Example:* Let us take the three sequences

A = s e n d e r

B = r e m i n d e r

C = s e l e c t e d

We first construct  $C' = s1 \ e1 \ l1 \ e2 \ c1 \ t1 \ e3 \ d1$ .

To construct  $A'$ , we take into account the symbols which are common to A and C, which are s, e, d. Only symbol e deserves some attention, because it is repeated in both A and C. The optimal matching between A and C corresponds to new sequence  $A' = s1 \ e1 \ n1 \ d1 \ e2 \ r1$ . This assignment leads to  $d(A, C) = d(A', C')$ .

We do the same for sequence B, obtaining  $B' = r1 \ e1 \ m1 \ i1 \ n1 \ d1 \ e3 \ r2$ , which leads to  $d(B, C) = d(B', C')$ . Obviously, we have  $d(A, B) \leq d(A', B')$ , because the optimal matching between A and B would correspond to  $A'' = s1 \ e1 \ n1 \ d1 \ e2 \ r2$  and  $B'' = r1 \ e1 \ m1 \ i1 \ n1 \ d1 \ e2 \ r2$ .



For the exact computation of  $d(A, B)$ , we have  $n_{AB} = n_B = 8$ . There are 3 non-common symbols (the **s** in ‘sender’, and the **m** and the **i** in ‘reminder’). They contribute with 3 to the final distance. We then have  $f(e, A, B) = 2/8$ ,  $f(n, A, B) = 2/8$ ,  $f(d, A, B) = 2/8$ ,  $f(r, A, B) = 1 + 2/8$ . Summing up, the distance is  $d(A, B) = 5$ .

On the other hand, one could analogously compute  $d(A, C) = 6 + 6/8$  and  $d(B, C) = 10 + 2/8$ . These three distances satisfy the three triangular inequalities.

#### IV. IMPLEMENTATION

In this section we show how the OSA distance can be efficiently computed.

First of all we need to determine which symbols are common or non-common to each sequence. This can be easily achieved using a vector of bits, one for each possible symbol of alphabet  $X$ . Initially the vector is full of zeros and, for each  $x \in A$ , its corresponding position in the vector is set to one. Now every symbol of  $B$  is tested against the vector. If the symbol was present in  $A$ , we increase the variable `common` that indicates the number of symbols in  $A \cap B$ . If the symbol was not present, then we increase a variable `non-commonB` that represents the number of symbols in  $B - A$ . The symmetric variable `non-commonA` is obtained as  $n_A - \text{common}$  after sequence  $B$  has been processed.

This allows us to compute the  $d(x, A, B)$  contributions, when  $x \notin X_A \cap X_B$ , in  $O(n_A + n_B)$  time and  $O(|X|)$  space. If  $|X|$  is very large and  $|X_A \cup X_B|$  is comparatively small, then the bit vector can be substituted by a hash table.

Finally, we have to explain how to compute the  $f(x, A, B)$  value that corresponds to  $d(x, A, B)$  for all  $x \in X_A \cap X_B$ . This task is more complex, since it involves the computation of the optimal assignment of symbol  $x$  positions, such that their global distance is minimum.

To solve this latter problem, we must store the positions of each common symbol. We adopt an even simpler global solution: we keep a vector of positions for each symbol (see Figure 1), not only the common ones. This allows us to reuse the auxiliary structure in subsequent comparisons (see Section IV-D). Using this structure, the non-common symbols are trivially found as before, since the position vector of a given symbol will be non-empty in one of the sequences and empty in the other. Moreover the structure allows solving the optimal alignment problem efficiently, as we explain in Section IV-A.

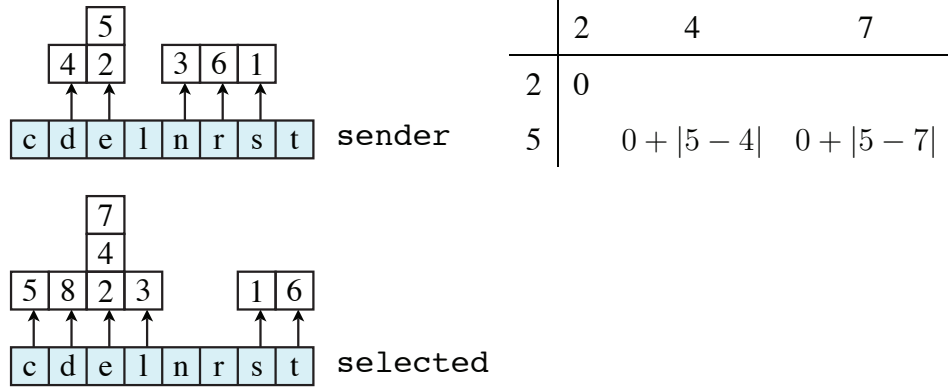


Fig. 1. Structures used to efficiently compute the distance between to sequences. (left) Vectors of positions of each symbol for the sequences **sender** and **selected**. (right) Matrix used to find the optimal alignment of the **e** symbols.

### A. The Optimal Symbol Alignment Algorithm

To solve the optimal symbol alignment problem of symbol  $x$ , we construct a matrix in which the positions of  $x$  in sequences  $A$  and  $B$  are used as row headers and the column headers, respectively. Without loss of generality, we assume that  $|A_x| \leq |B_x|$ . Each position of symbol  $x$  in  $A$  is assigned to one row. Using the notation introduced in Section III, we assign position  $i_p$  to row  $p$ . Note that positions are ordered. The same is done for each position of  $x$  in  $B$  and each column, so that position  $j_q$  is assigned to column  $q$ . For instance in Figure 1 we can see a matrix corresponding to the symbol **e** in the sequences **sender** and **selected**.

We follow a dynamic programming strategy: the matrix is computed by rows, and each cell  $(p, q)$  contains the minimum cost of all previous cells of preceding row, plus the cost of assigning occurrence  $p$  of symbol  $x$  in  $A$  to occurrence  $q$  of  $x$  in  $B$ , i.e.  $|i_p - j_q|$ . Formally, the content of the matrix is defined recursively as follows:

$$\text{cost}(p, q) = \min_{k=p-1, \dots, q-1} \{\text{cost}(p-1, k)\} + |i_p - j_q|$$

The base case is defined by the cells of the first row, that simply contain the cost of the assignment:  $\text{cost}(1, q) = |i_1 - j_q|$ . To simplify the notation, we define the function  $\text{mc}(p, q)$  as  $\min_{k=p, \dots, q} \{\text{cost}(p, k)\}$ , thus  $\text{cost}(p, q) = \text{mc}(p-1, q-1) + |i_p - j_q|$ .

Since an occurrence can only be assigned once, assignments in the optimal alignment respect the order of the positions, i.e.  $i_{p_1} < i_{p_2} \Rightarrow \text{pj}(i_{p_1}, A, B) < \text{pj}(i_{p_2}, A, B)$ . Furthermore, enough occurrences must be left for the remaining symbols not already assigned. Therefore, only  $|B_x| -$

$|A_x| + 1$  cells of each row have to be filled. For row  $p$ , the first column to compute is  $p$  and the last one is  $|B_x| - |A_x| + p$ . However there is an additional factor that usually cuts down the number of cells that have to be computed. Since in the computation of a row only the minimum value of previous cells is used, and this value becomes steady after the minimal value of the row is achieved, then we need only to compute a row until its minimum value is found.

For a fixed row  $p$ , the increase/decrease of the  $\text{cost}(p, q)$  function in terms of the column  $q$  can be studied from the behavior of both the  $\text{mc}(p-1, q-1)$  and  $|i_p - j_q|$  functions. As said before, the  $\text{mc}$  function is always decreasing, until it stabilizes. On the other hand, the value of  $|i_p - j_q|$  decreases, achieves its minimum point, and then increases (although only one of these two behaviors is observed if there are restrictions caused by previous assignments and/or the number of remaining positions to assign). The minimum value of the  $\text{cost}(p, q)$  function must be between the two minimum points of its component functions. Thus, for a particular row  $p$  we only have to check until column  $\min\{|B_x| - |A_x| + p, \max\{q_r^{p-1} + 1, q_s^p\}\}$ , where  $q_r^{p-1}$  is the first column where the value of  $\text{cost}(p-1, q_r^{p-1})$  is the smallest in the  $p-1$  row, and  $q_s^p$  is the column in which  $|i_p - j_{q_s^p}|$  is minimal (that is, if  $i_p$  corresponded to the only symbol  $x$  in  $A$ , then  $q_s^p$  would be the optimal placement for it, in  $B$ ).

Algorithm 1 describes the algorithm to compute the optimal alignment for a symbol  $x \in A_x \cap B_x$ . Since the matrix is computed by rows and only the values of previous row are needed, only two rows are stored at any time; thus the space complexity of the algorithm is  $O(2 \cdot |B_x|)$ . The variables `prev_row` and `curr_row` are pointers to the previous row values and current row values, respectively. Their smallest values are kept in variables `prev_small` and `curr_small`. Note that `prev_small` is used not to maintain the absolute smallest value, but the smallest observed value up to the current column, i.e. the value of  $\text{mc}(p-1, q-1)$ .

The value of  $q_r^{p-1}$  is used for two purposes: to keep updating `prev_small` until the column in which the last value in previous row was computed, and to determine if we can finish the processing of current row. As we have seen before, we can stop when current column is past  $q_r^{p-1}$ , i.e.  $q > q_r^{p-1}$ , and when the value of  $|i_p - j_q|$  is minimal. Since, for a given row,  $i_p$  is a constant value, the minimal point of the absolute value corresponds to the point where the difference between  $i_p$  and  $j_q$  is around 0. Thus we must only look until the first value  $j_q$  which is larger or equal than  $i_p$ .

---

**Algorithm 1:** Optimal Alignment

---

```

 $q_r^{p-1} \leftarrow 0$  ▷ Column of previous row with smallest value
prev_small  $\leftarrow 0$  ▷ Previous row smallest value before column  $q$ 
for  $p = 1$  to  $|A_x|$  do
    curr_small  $\leftarrow \infty$  ▷ Current row smallest value
    for  $q = p$  to  $|B_x| - |A_x| + p$  do
        if  $1 < q \leq q_r^{p-1} + 1$  then
            prev_small  $\leftarrow \min(\text{prev\_small}, \text{prev\_row}[q - 1])$ 
            curr_row[ $q$ ]  $\leftarrow \text{prev\_small} + |i_p - j_q|$ 
            if curr_small  $>$  curr_row[ $q$ ] then
                curr_small  $\leftarrow \text{curr\_row}[q]$ 
                 $q_r^p \leftarrow q$ 
            if  $(j_q \geq i_p)$  and  $(q > q_r^{p-1})$  then break
        swap (prev_row, curr_row)
     $q_r^{p-1} \leftarrow q_r^p$ 
    prev_small  $\leftarrow \infty$ 
return prev_row[ $q_r^{p-1}$ ]
    
```

---

*B. A More Complex Example*

Figure 1 exemplifies the behavior of the algorithm on our running example of Section III-A1. However, because there are few repeated symbols in the considered strings, it does not allow to fully appreciate many of the aspects that must be taken into account. Here we present a more illustrative example.

Let us take strings  $A = \text{bbbbbbbbbaaa}$  and  $B = \text{aaacccccccaccccaaccccccccccca}$ . There are two non-common symbols, **b** and **c**, that contribute 9 and 20, respectively, to the distance. Now we pay attention to the optimal symbol alignment of the common symbol **a**. The following table is the outcome of the algorithm.

	1	2	3	10	15	16	27
10	0+9	0+8	0+7	<b>0 + 0</b>			
11		9+9	8+8	7 + <u>1</u>	<b>0+4</b>		
12			18+9	16 + <u>2</u>	8+3	<b>4+4</b>	

Each cell contains the sum of the smallest value of all previous columns of previous row, plus the cost of the current assignment (in that order). The smallest row value is marked in bold font, if current row is  $p$  this marks the column  $q_r^p$ . The smallest  $|i_p - j_q|$  value of each row  $p$  is underlined, its column corresponds to  $q_s^p$ . For all three symbols  $a$  of sequence  $A$ , its optimal assignment is with the symbol in position 10 of  $B$  (column 4). For the first row, the optimal assignment is the one that limits the number of cells to be computed, but for the other two rows, the threshold is provided by the column containing the smallest value of previous row. In this example three cells are saved in total with respect to the cells that would have been computed without the cut-off rules.

### C. Algorithm Complexity

In terms of time complexity, Algorithm 1 is composed of two nested loops in which all the operations take constant time (the `swap` operation simply exchanges the pointers of both rows). Thus the worst time complexity of the algorithm is  $O(|A_x| \cdot (|B_x| - |A_x| + 1))$ . However, the bounds provided by the minimal cell value usually save the computation of most of the cells, and our experimental results reveal that the computation of the optimal alignment for all common symbols takes the same time as constructing the vector positions. Since this latter operation takes linear time, this means that the optimal alignment exhibits a similar behavior in practice (for our particular setting).

The time complexity of the optimal alignment allows us to obtain the complete cost of computing the OSA distance, which is

$$O(n_A + n_B + \sum_{x \in A_x \cap B_x} |A_x| \cdot (|B_x| - |A_x| + 1)).$$

Thus, computation time ranges from  $O(n_A + n_B)$ , if no symbol is shared between  $A$  and  $B$ , to  $O(n_A + n_B + n_A \cdot (n_B - n_A + 1))$  if all symbols are equal. This latter formula is obtained disregarding the cut-off rule of the minimal cell value. The impact of this rule is non-negligible: for instance, if all symbols are equal, our algorithms actually takes  $O(2n_A + n_B)$ , assuming  $n_A \leq n_B$ , because each symbol in  $A$  finds its optimal position in the first cell it computes.

### D. Additional Advantages

Many of the existing distances build auxiliary structures that are only useful when comparing two particular sequences  $A$  and  $B$ . Thus they cannot be reused in subsequent comparisons, even

if only one of the sequences is different (e.g. when comparing  $A$  and  $C$ ). Our implementation can save the construction of the position vector of the sequence that does not change, obtaining faster processing in some comparison-intensive scenarios like record linkage.

### E. An Alternative Version

The algorithm we have provided is practical, but still computes many unnecessary cells. Since the minimum value of the function  $\text{cost}(p, q)$  must be between the two minimum points of its component functions, we can skip the computation of some of the first columns. In fact, for row  $p$ , we need only to compute the cell values in the columns in the interval

$$[\max\{p, \min\{q_r^{p-1} + 1, q_s^p\}\}, \min\{|B_x| - |A_x| + p, \max\{q_r^{p-1} + 1, q_s^p\}\}].$$

Actually this interval can be further refined, since the min function decreases and then becomes steady. If  $q_s^p \geq q_r^{p-1} + 1$ , then the column with smallest value in row  $p$  is column  $q_s^p$ . Only in the other case we have to search the interval. Thus

$$\begin{cases} q_r^p = \min(|B_x| - |A_x| + p, q_s^p) & \text{if } q_s^p \geq q_r^{p-1} + 1 \\ q_r^p \in [\max(q_s^p, p), q_r^{p-1} + 1] & \text{otherwise.} \end{cases} \quad (1)$$

The problem with this approach is that, although the value  $q_r^{p-1} + 1$  is already available when we start the processing of row  $p$ , the value  $q_s^p$  is not known beforehand. However, there is a simple alternative to the linear scan to find out its value: a binary search can be performed to find its optimal assignment column, since the column values are ordered. Moreover, since rows are also ordered, the range in which the binary search has to be performed for row  $p$  is  $[q_s^{p-1}, |B_x| - |A_x| + p]$ .

Using this alternative version, the table computed in the example presented in Section IV-B is reduced to

	1	2	3	10	15	16	27
10				<b>0 + 0</b>			
11				<b>7 + 1</b>	<b>0+4</b>		
12				<b>16 + 2</b>	<b>8+3</b>	<b>4+4</b>	

An immediate observation is that the table is, in fact, incomplete, since the values 7 and 16 can only be obtained by computing the value of cells that do not appear in the table. However,

the values to be computed either were already computed in previous steps or depend simply on  $\text{cost}(p-1, q-1)$ , as the following proposition states.

*Proposition 2:* The value  $\text{cost}(p, q)$  depends only on already computed values (previous row) or on the value  $\text{cost}(p-1, q-1)$ .

*Proof:* Computing  $\text{cost}(p, q) = \text{mc}(p-1, q-1) + |i_p - j_q|$  reduces to computing  $\text{mc}(p-1, q-1)$ , since  $|i_p - j_q|$  can be obtained in constant time. Consider the behavior of the  $\text{mc}$  and  $\text{cost}$  functions in row  $p-1$ . If  $q_s^{p-1} \geq q_r^{p-2} + 1$ , then  $\forall q \leq q_s^{p-1}, \text{mc}(p-1, q) = \text{cost}(p-1, q)$ , and  $\forall q > q_s^{p-1}, \text{mc}(p-1, q) = \text{cost}(p-1, q_s^{p-1})$ . In this case  $q_r^{p-1} = q_s^{p-1}$  and the values of  $\text{cost}(p-1, q)$  decrease until column  $q_s^{p-1}$  is reached. Thus  $\text{mc}$  mimics this behavior and then in subsequent columns its value remains stable.

On the other hand, if  $q_s^{p-1} < q_r^{p-2} + 1$ , then there is an interval of columns in which the minimum value  $q_r^{p-1}$  can appear, namely  $[\max(q_s^{p-1}, p), q_r^{p-2} + 1]$ . Precisely these are the columns computed by the algorithm. For columns outside this interval, there are two possibilities: if column  $q$  is on the right of the interval ( $q > q_r^{p-2} + 1$ ), then  $\text{mc}(p-1, q) = \text{cost}(p-1, q_r^{p-1})$ , which was already computed; if  $q$  is on the left of the interval ( $q < \max(q_s^{p-1}, p)$ ), then in this zone the values of the cost function are always decreasing, thus  $\text{mc}(p-1, q) = \text{cost}(p-1, q)$ . ■

As a consequence of this proposition, if cells are missing in our table when we try to compute the cells required by Equation 1, the missing cells can be only the ones in the diagonals. This can be observed in our example, since the values 7 and 16 can be obtained by computing the missing diagonals. Thus it is possible to compute the lower bound of each row before computing any cell value.

First of all we compute all  $q_s^p$  values doing a binary search in the range  $[q_s^{p-1}, |B_x| - |A_x| + p]$  and we store them in a vector. Then we detect blocks of identical  $q_s^p$  values doing a linear scan. These are the only cases in which non-previously computed cells have to be considered. Every time a block with more than one element is found, we update the lower bounds of previous rows considering the diagonal needed by the block. The update is stopped whenever a row had an already smaller lower bound. For instance, in our example there is a single block of three elements with  $q_s^1 = q_s^2 = q_s^3 = 4$ . The diagonal requires that first column to be computed in row 2 is column 3, while in the first row we start from column 2.

Once lower bounds are available, we use a variation of Algorithm 1 in which we only scan

the columns from the lower bound of the row to the upper bound determined by  $\min(|B_x| - |A_x| + p, \max(q_r^{p-1} + 1, q_s^p))$ . Since the last cell to be computed in the row is already known, the last if command is useless and can be suppressed.

This alternative version of the algorithm computes always less cells than Algorithm 1, using only two additional vectors to store the lower bounds and the  $q_s^p$  values. However, when compared to Algorithm 1, the performance of the alternative version is better only in some particular scenarios. In general, for strings in which repetitions are not very numerous, the logarithmic factor of the binary search is not so advantageous when compared to the linear scan. In such cases the regularity of memory access of the linear scan can give better performance than the random memory access pattern of the binary search approach, specially if we consider that there are also some additional operations to be performed, like the detection of blocks and the update of lower bounds.

On the other hand this alternative version is very suited for scenarios in which there are lots of symbol repetitions and the number of elements that share optimal position is low (*i.e.* the size of the blocks is small). In Section V-B we will see that this is indeed the case for long DNA sequences.

In next section, we will refer to this alternative algorithm as OSA<sub>2</sub>, whereas we will denote Algorithm 1 as OSA<sub>1</sub>. We stress that both algorithms output the same value for the OSA distance; therefore, there are no quality differences between using Algorithm OSA<sub>1</sub> or OSA<sub>2</sub>. The only measurable difference between them is their execution time.

## V. EXPERIMENTS AND APPLICABILITY

In this section, we describe the experiments that we have carried out to test the performance (running time and quality results) of OSA distance. The general conclusion that we draw from these experiments is that OSA distance is a perfect candidate to be used in situations requiring intensive comparisons of sequences of symbols.

First we describe the datasets that we have considered for our experiments. Then we explain a first experiment dealing with very long DNA sequences. After that, we detail some experiments for the record linkage problem. Finally, we explain other situations where the OSA distance could be used.



### A. Database Generation and Computational Environment

We have basically run two kinds of experiments: computation of distances between long DNA sequences, and record linkage methods. All the experiments have been performed in an Intel Core2 64 bits CPU (2.13GHz) with 2Gb of RAM memory. We have used C++ compiled with *gcc* 4.1.2 to implement all the aforementioned distances and algorithms, as well as, the record linkage software needed for performing the experiments described later on.

For the experiment related to DNA sequences, we compute the (Edit, OSA, Jaro-Winkler) distance between the complete genome of the *Saccharomyces cerevisiae* yeast (the mitochondrial genome was taken as one sequence, as well as each one of the yeast chromosomes, for a total of 17 sequences), and the mitochondrial genomes of 129 species [20]. The total number of bases in the *Saccharomyces cerevisiae* genome is over 12 million, the longest sequence containing about 1.5 million bases. In the other database there are 3M bases, the longest sequence being 0.2 million bases long.

For the experiments related to record linkage, we have considered databases with two sizes: 25,000 entries (medium size) and 1 million entries (large size). In order to create a realistic record linkage scenario, we have used names and surnames extracted from a frequency dictionary containing 1,564 names and 13,068 surnames obtained from the Catalan Official Statistics Institute (IDESCAT) [21]. We have generated one database (*Dm*) containing 25,000 different full names, and another database (*DI*) containing 1 million full names. The maximum length of the generated records is 44 symbols. For each of these two original databases, we have generated three duplicated databases, each one corresponding to a different scenario (easy, normal and hard) of insertion of errors. The first one simulates the scenario where names are manually added in the database; in this case, errors are produced by typos or misspellings. The second one simulates the OCR (optical character recognition) scenario where the amount of mistakes is usually larger. Finally, the latter simulates a hard scenario where strings have a high complexity, as for instance, in the Dutch census.

The duplicated databases are generated by perturbing each entry of the original database, according to different distributions of the mistakes. For the former case (manual scenario), we have taken the number of errors following a uniform distribution between 1% and 5% of the length of each full name. For the second case (OCR scenario), we follow a uniform distribution

between 1% and 15% of the length of each full name. Finally, for the latter case (Dutch scenario), we follow a uniform distribution between 1% and 40% of the length of each full name. The considered errors are the following ones: insertions, deletions, updates and swaps, all of them with the same probability of occurrence. The election of these distributions is based on the study developed in [22]. In such study, the author shows that manual typed texts have few mistakes due to typing errors (1-3.2%) and spelling (1.5-2.5%) errors, i.e. around 5% in total. Whereas, text collections digitalized via OCR contain a percentage of errors around 15%. However, as it is also described in this study, in certain scenarios, like typing Dutch surnames (by Dutch), the error rate reaches 38%; this latter case corresponds to our hard scenario.

Summing up the generation of databases for our record linkage experiments, we have two original databases  $Dm$  and  $Dl$ , then three duplicated versions  $Dm_1, Dm_2, Dm_3$  of  $Dm$ , each one with 25,000 entries, and finally three duplicated versions  $Dl_1, Dl_2, Dl_3$  of  $Dl$ , each one with 1 million entries.

The two kinds of experiments, with DNA sequences on the one hand and sequences of name and surname on the other hand, provide results for two different situations: long sequences with a short alphabet of 4 symbols (in the DNA case) and short sentences with a larger alphabet of 33 symbols (in the record linkage case).

### *B. Long DNA Sequences*

Our first experiment has been thought just to compare the running time of the three considered distances in scenarios with very long sequences of symbols. Specifically, we have computed the distance between the complete genome of the *Saccharomyces cerevisiae* yeast and the mitochondrial genomes of 129 other species. This experiment involves therefore sequences which are millions of symbols long.

Table I shows the time that was necessary to compute each of the three distances. The times for Edit and Jaro-Winkler distances are very high. This is not surprising at all, taking into account that the cost of computing the Edit distance is quadratic in the number of symbols, and that the Jaro-Winkler distance was designed to be more efficient for not too long sequences of symbols.

Regarding the two implementations of the OSA distance, the results of this experiment confirm that the alternative implementation  $OSA_2$  is faster than  $OSA_1$  when the number of symbol

Distance	Time (seconds)
Edit	230724.0 (2.67 days)
Jaro-Winkler	31513.8 (8.75 hours)
OSA <sub>1</sub>	92.0
OSA <sub>2</sub>	12.5

TABLE I  
 RUNNING TIME TO COMPUTE DNA DISTANCES.

repetitions in a sequence is large. We believe that the two proposed implementations of the OSA distance are of independent interest.

Note that we have not considered in this experiment any formal quality measure for the behavior of the distances. This will be done in the next section, devoted to record linkage. However, just to confirm our intuition that OSA distance and Edit distance behave quite similarly, we have graphically represented the relation between OSA distance and Edit distance of all pairs of DNA sequences that have been compared in this experiment, in Figure 2. Namely, for each pair  $s_1, s_2$  of DNA sequences that have been compared, the point  $(d_{\text{Edit}}(s_1, s_2), d_{\text{OSA}}(s_1, s_2))$  is added to the graph.

### C. Record Linkage

Record Linkage is a technique widely used for data cleaning [23] and integration of distributed and non-homogeneous databases [24]. Typically, such databases contain information (records) about common individuals that, frequently, do not match due to errors in the data. These errors can be accidentally produced (e.g. typos or misspelling errors) or intentionally provoked (e.g. data anonymization).

The goal of record linkage is therefore to compare two databases  $X, Y$  and find pairs of records (one in  $X$ , one in  $Y$ ) which correspond to the same individual. A very common approach to this problem is distance-based record linkage: for each record  $a$  in one of the databases, one searches the record(s) in the other database that is (are) at minimum distance to  $a$ , for some distance defined on the domain of the records. More formally, for all  $a \in X$ , the subset

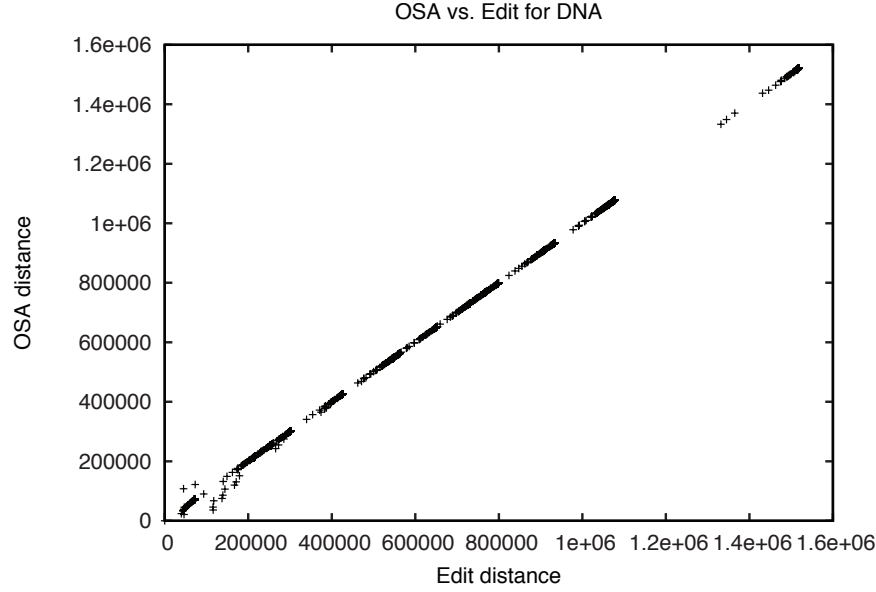


Fig. 2. Correlation between Edit distance and OSA distance in the DNA sequences database.

$Y_a = \{b' \in Y \text{ s.t. } d(a, b') \leq d(a, b), \forall b \in B\}$  is found. Here  $d(a, b)$  is a distance between a record of the database  $X$  and a record of the database  $Y$ .

In our first experiments, we start with an original dataset  $X = Dm, Dl$ , then we consider as the second database  $Y$  one of the perturbed versions of  $X$ . We denote as  $\psi$  the perturbation function that has been used to generate  $Y$ , and we denote as  $a' = \psi(a)$  the elements in  $Y$ , for all  $a \in X$ . We apply some record linkage algorithm to the pair of databases  $(X, Y)$ , by considering as the distance  $d(a, b)$  either the Edit distance, or the Jaro-Winkler distance, or the OSA distance. Apart from computing the global running time of this record linkage process, the idea is to measure its quality. This is done by computing the fraction of records  $a \in X$  such that the valid perturbed record  $a' \in Y$  (that we control, because we have generated it) belongs to the subset  $Y_a$  of records at minimum distance to  $a$ . This percentage is denoted as the *recall* of the record linkage process, which measures the fraction between the number of true positives and the total number of real positives. In our case,

$$\text{recall} = \frac{|\{a \in X \text{ s.t. } a' \in Y_a\}|}{|X|}$$

The cardinality of the subset  $Y_a$  is also a quality measure of the record linkage process: assuming that  $a' \in Y_a$ , the smaller  $Y_a$  is, the more successful and efficient the record linkage

process is. This quality measure is denoted as the *precision* of the record linkage process; it measures the fraction between the number of true positives and the total number of labeled positives. In our case,

$$\text{precision} = \frac{|\{a \in X \text{ s.t. } a' \in Y_a\}|}{\sum_{a \in X} |Y_a|}$$

Finally, a measure that combines precision and recall is the *F-measure*, which is the harmonic mean of precision and recall:

$$\text{F-measure} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

In the experiments described in the next sections, we have computed the total running time, the recall, precision and F-measure of each process. We have implemented three different versions of the Edit distance: the classical one with a complexity equal to  $O(n^2)$ , an alternative version that stores only two rows at a time of the dynamic programming table, and the variant presented in [19], with a complexity equal to  $O(dn)$ , where  $d$  is the number of edits for converting one string into the other. The time value presented in the tables is the smallest one. For the Jaro-Winkler distance we have used the implementation provided by the authors, publicly available in [25]. For the OSA distance we have implemented both the alignment algorithm  $\text{OSA}_1$  presented in Section IV-A and its alternative version  $\text{OSA}_2$  of Section IV-E. In all cases the implementations were modified to be as efficient as possible for the record linkage process. That is, memory was allocated only at the beginning; and as many of the auxiliary structures created in previous comparisons as possible were reused.

1) *Experiments with Medium Databases:* First of all we consider  $X = Dm$  as the first dataset, and  $Y = Dm_1, Dm_2, Dm_3$  as the second database. Since the sizes of these datasets (25,000 records) are medium, we can consider a naive implementation of record linkage (nested loop join) where the element  $b' = \arg\min_{b \in Y} d(a, b)$  is found by computing  $d(a, b)$  for all  $b \in Y$ . This is repeated for all the elements  $a \in X$ . Note that this means  $(25,000)^2 = 625$  millions distance computations.

Tables II,III,IV show the results obtained for this record linkage experiment involving medium-sized databases. We can draw the following conclusions from the results in the tables.

- As expected, the OSA distance is always the fastest distance, three times faster than the Edit distance and two times faster than the Jaro-Winkler one. Although  $\text{OSA}_2$  is slower than  $\text{OSA}_1$ , it still beats the two other distances.

Distance	Time (s)	Recall	Precision	F-measure
Edit	1711.11	0.99996	0.9994	0.99968
Jaro-Winkler	1239.88	0.97412	0.973536	0.973828
OSA <sub>1</sub>	530.14	1	1	1
OSA <sub>2</sub>	774.63	1	1	1

TABLE II  
 RESULTS FOR THE MANUAL SCENARIO,  $X = Dm$  AND  $Y = Dm_1$ .

Distance	Time (s)	Recall	Precision	F-measure
Edit	1712.49	0.99924	0.994704	0.996967
Jaro-Winkler	1248.77	0.82936	0.826023	0.827688
OSA <sub>1</sub>	530.05	0.99452	0.994281	0.9944
OSA <sub>2</sub>	796.46	0.99452	0.994281	0.9944

TABLE III  
 RESULTS FOR THE OCR SCENARIO,  $X = Dm$  AND  $Y = Dm_2$ .

- Regarding the quality results, the OSA distance obtains the best results in the manual scenario ( $Dm_1$ ). The results achieved by OSA distance are very similar to the ones obtained by Edit distance in the OCR scenario ( $Dm_2$ ). Note that these two are the most common scenarios in record linkage, where such techniques are used to fix the problems of hand-made or OCR database entries. Only in the most difficult scenario (the rare and extreme Dutch one, corresponding to database  $Dm_3$ ) OSA distance obtains really worse results than Edit distance.

We would like to emphasize that the quality values obtained by the OSA distance are always better than the ones obtained by Jaro-Winkler distance, which is commonly used in record linkages scenarios.

- Summing up, in very hard scenarios where duplicates are very distant from originals, Edit distance is the best choice. For other (maybe more realistic) scenarios, the OSA distance achieves similar or better quality results than Edit distance, but using OSA is three times

Distance	Time (s)	Recall	Precision	F-measure
Edit	1715.22	0.91616	0.722205	0.807702
Jaro-Winkler	1261.39	0.33864	0.334969	0.336794
OSA <sub>1</sub>	533.08	0.547	0.538259	0.542594
OSA <sub>2</sub>	772.66	0.547	0.538259	0.542594

TABLE IV  
 RESULTS FOR THE DUTCH SCENARIO,  $X = Dm$  AND  $Y = Dm_3$ .

faster than using Edit. In some cases, Edit distance leads to better recall, whereas OSA distance leads to better precision and F-measure. This happens because it is more likely that more than one record in  $Y$  achieves the minimum Edit-distance to an original record  $a$ , whereas repetitions of OSA distances are more unlikely. Therefore, the cardinal of the set  $Y_a$  that contains the perturbed records that achieve the minimum distance to the original record  $a$  is usually bigger in the Edit distance case than in the OSA distance case.

2) *Experiments with Large Databases:* Now we consider the large databases:  $X = Dl$  as the first dataset, and  $Y = Dl_1, Dl_2, Dl_3$  as the second database. This means that the two databases contain 1 million records each, and so the naive nested loop join approach would be completely unfeasible in this case. Also, in order to make this problem even more difficult, we consider one unique database  $Z = X \cup Y$  where  $Y = Dl_1, Dl_2, Dl_3$ . Now, the problem we want to solve is a self join problem instead of a nested one. Usually, this is the kind of problems that cleansing applications face in real scenarios: to find duplicates in a single very large database [26], [27].

Different techniques for saving distance computations in record linkage have been presented in the literature. The main idea is to apply some filtering or partitioning criterion in such a way that very different strings do not need to be compared. Classical methods include *blocking* and *sliding window* techniques [28], [29]. The former defines a blocking key and only the records (or strings) sharing such key are compared. The latter sorts the strings using a sorting criterion and then only the strings placed inside a window of a predefined size are compared. The window is normally centered on the record to be linked. These two techniques are usually repeated several times by changing the blocking or sorting criteria.

Recently, more sophisticated filtering and partitioning methods have been proposed, which use information on the prefixes, lengths,  $q$ -grams, etc. of the compared strings. For instance, in [30] authors map the strings into an Euclidean space in order to later perform all the distance computations in this new space; this strategy avoids the costly calculation of many Edit distances. A different approach is proposed in [31], where the matching condition is relaxed to reduce the number of comparisons to be done. A very recent filtering technique, called *ed-join*, has been presented in [32]. Ed-join uses the  $q$ -grams information to discard strings that are a priori too far to form a correct linkage with the considered record. We stress that these methods are designed to solve a different record linkage problem than the one we consider here: similarity joins with respect to a given threshold  $k$ . That is, the goal is to find all the pairs of records whose distance is less than  $k$ . The above-cited filtering methods, which are specific for the Edit distance, solve this problem without any loss of recall: all the pairs at Edit distance less than  $k$  are found. However, in the record linkage problem that we consider here, where the goal is to find the perturbed record that corresponds to a given original record, even these methods have a loss of recall, because maybe the closest record (with respect to Edit) is not the correct answer. Note that recall has two different meanings, in these two record linkage problems.

In the above-mentioned record linkage protocols, the total execution time can be divided into two parts: (1) the time required to set up and apply the filtering conditions ( $q$ -grams representation, string sorting, etc.), which leads to a set of candidates  $Z_a$  for each record  $a \in Z$  to be linked; and (2) the cost of computing the distances  $d(a, b)$ , for the candidates  $b \in Z_a$ , in order to obtain the record(s) which is/are closest to each  $a$ . Therefore, in principle there is a trade-off between these two running times: if the process spends more time in filtering distant records, then less distances  $d(a, b)$  will be computed.

This clearly shows that significant differences between the execution times of two different distances (such as Edit and OSA) can have more or less impact on filtering-based record linkage protocols, depending on the time employed in each of the two parts of the linkage process. For example, if the employed filtering technique is very accurate, then most of the global running time of the record linkage process will be devoted to filtering. In this case, there will be no significant difference between using Edit or OSA when computing actual distances (although employing OSA will be always faster). However, if a more naive, simpler and faster filtering technique such as sliding window is employed, then a lot of distance computations will be necessary in



the second phase of the record linkage process. In this case, there will be significant differences when using Edit or OSA distance.

We describe now a set of self-join record linkage experiments that we have executed to discuss this issue. For this new set of experiments, we have discarded the Jaro-Winkler distance because the results obtained in the previous experiments (in an easier scenario) are clearly worse than the ones obtained by Edit and OSA distances. For the OSA distance, we have considered only the first algorithm  $OSA_1$  because the alternative one  $OSA_2$  is slower for short strings.

On the one hand, we have implemented the classical sliding window algorithm, with both Edit and OSA distances for the second phase, and we have executed such algorithm on the databases  $Z = X \cup Y$  where  $Y = Dl_1, Dl_2, Dl_3$ . The window size  $w$  has been chosen ad-hoc in such a way that the record linkage process has a similar or smaller running time than record linkage with ed-join filtering. We have applied four sorting criteria: lexicographical order for the names and for the reversed names, and lexicographical order for the surnames and for the reversed surnames. A record is compared to all the records that are inside some of the four resulting sliding windows. An execution of this record linkage protocol will be denoted as  $SlidingWindow(w) + \text{Edit or OSA}$ , in Tables V,VI,VII.

On the other hand, we have also executed the ed-join record linkage algorithm from [32]<sup>1</sup>, using Edit distance for the second phase. The ed-join algorithm has two different parameters. The first one,  $q$ , refers to the size of the  $q$ -grams considered for the filtering; this parameter usually ranges from 2 to 5. The second parameter,  $t$ , refers to the threshold for the distance between two non-filtered records: the set of candidates  $Z_a$  will contain all the records at distance less or equal than  $t$  from the corresponding record  $a$ . Depending on the values of  $q$  and  $t$ , the ed-join algorithm discards a large number of records that can have an edit distance lower than  $t$  (so they should be checked afterwards), because some of the implemented filters require that the strings have a size larger than  $q \cdot (t + 1)$  to correctly operate; in these cases, the authors of [32] recommend to repeat the ed-join filtering for these records with a smaller value of  $q$ , to achieve a better recall. For this reason, an execution of this ed-join record linkage protocol will be denoted in Tables V,VI,VII as  $\text{Ed-Join}(q, t)$  if only a value of  $q$  is considered, or as  $\text{Ed-Join}((q_1, q_2), t)$  if two values of  $q$  are considered.

<sup>1</sup>the original implementation can be downloaded from the web site: <http://www.cse.unsw.edu.au/~weiw/project/simjoin.html>

Record Linkage method	Time (s)	Recall	Precision	F-measure
Ed-Join(2,2)	4282	0.998728	0.987995	0.993333
Ed-Join(5,2)	489	0.99263	0.987923	0.990271
SlidingWindow(40) + Edit	655	0.975124	0.933696	0.953960
SlidingWindow(70) + Edit	1124	0.980173	0.943743	0.961613
SlidingWindow(40) + OSA	249	0.973627	0.96814	0.970875
SlidingWindow(70) + OSA	410	0.978628	0.97400	0.976309

TABLE V  
 RESULTS FOR THE MANUAL SCENARIO,  $Z = Dl \cup Dl_1$ .

Tables V,VI,VII contain the results of these experiments: running time, recall, precision and F-measure. We discuss now the most relevant conclusions that can be drawn from the obtained results.

- Ed-join obtains (slightly) better quality results than sliding window for the easy manual scenario. However, for the other two scenarios, OCR and Dutch, sliding window obtains better quality results, with (much) lower running times. This happens because ed-join is designed for scenarios with low error rate and large alphabets. As stated in [32] larger values of  $q$  tend to yield better running times, at the price of augmenting the number of non-processed strings, thus decreasing the final recall.
- Focusing on the sliding window method in order to compare the performance of OSA and Edit distances, the first obvious consequence is the different running times: employing OSA is almost three times faster than employing Edit, for the same size of the window. This allows us to consider bigger windows with the OSA distance (from 40 to 70, or from 1000 to 2000), to improve the quality results obtained by Edit, still with a lower running time.
- Even if sliding window + Edit may lead to better recall results than sliding window + OSA (not in the Dutch case, though), the obtained precision when using Edit is significantly worse. This is due to the fact that the Edit distance has few possible values (non-negative integers), whereas the OSA distance has much more possible values. Therefore, it is more likely that different strings are at the same (minimum) Edit distance of a considered string, which leads to poorer precision results. The consequence is that the global quality parameter

Record Linkage method	Time (s)	Recall	Precision	F-measure
Ed-Join(3,3)	2513	0.632738	0.810423	0.710642
Ed-Join(3,4)	9630	0.851479	0.788442	0.818749
Ed-Join(4,4)	5520	0.500562	0.759478	0.603418
Ed-Join((4,3),4)	12407	0.85148	0.788505	0.818783
SlidingWindow(1000) + Edit	17015	0.890344	0.708949	0.789359
SlidingWindow(1000) + OSA	6393	0.85481	0.84937	0.852081
SlidingWindow(2000) + OSA	12656	0.87288	0.867595	0.870229

TABLE VI  
 RESULTS FOR THE OCR SCENARIO,  $Z = Dl \cup Dl_2$ .

Record Linkage method	Time (s)	Recall	Precision	F-measure
Ed-Join(3,4)	5740	0.013638	0.029988	0.018749
Ed-Join(3,5)	15037	0.023763	0.0410613	0.030104
Ed-Join((3,2),5)	25902	0.053829	0.070462	0.061032
Ed-Join((4,3),5)	16204	0.023763	0.041061	0.030104
SlidingWindow(1000) + Edit	17124	0.293453	0.128245	0.178487
SlidingWindow(1000) + OSA	6361	0.205773	0.200904	0.2033096
SlidingWindow(2000) + OSA	12766	0.224815	0.219351	0.2220494

TABLE VII  
 RESULTS FOR THE DUTCH SCENARIO,  $Z = Dl \cup Dl_3$ .

of F-measure obtained by using OSA is better.

- Summing up, our experiments show that a record linkage protocol where the filtering phase is more simple (such as sliding window) leads in some cases to better results than a more complicated record linkage process such as ed-join, especially if the distance employed in the second phase of the process is faster.

Besides the main conclusions, a number of additional interesting results arise from these experiments:

- Comparing the performance of ed-join in the experiments of [32] and the one obtained in our setting, ed-join executes much slower with our benchmarks. For instance Xiao et al. achieve

running times in the order of seconds for a dataset containing 863,171 records. However the maximum value they consider for  $t$  is 3, the average length of the strings is 104.8 (which allows to use 5-grams as a filtering criterion without discarding too many strings), and the alphabet contains 93 symbols, which makes it more likely to obtain non-frequent  $q$ -grams that are very useful to prune candidates. In contrast, our benchmark contains more strings but they are shorter, which precludes the use of large values for  $q$ , and the alphabet is smaller. Consequently, there are fewer non-frequent  $q$ -grams and the pruning power of some of the filters is severely diminished, resulting in more comparisons to be done and, so, higher execution times.

- In the Dutch scenario there is a noticeable difference between the recall values obtained with large databases when compared to the experiments with medium-sized databases. To explain these differences, let us recall that the two experiments are different: in the experiments with medium-sized databases the (nested-loop) record linkage process is executed between an original database  $X$  and a perturbed database  $Y$ , whereas in the experiments with large databases, the record linkage problem is a self join one, executed in a global database  $Z = X \cup Y$  containing both original and perturbed records. In the Dutch scenario for large databases, there are a lot of strings and the distances between original and perturbed strings are quite big, so the true duplicate of an original record is usually more far away than some other original record. Furthermore, since we use filtering techniques in the experiments with large datasets (and not in the experiments with medium-sized datasets), it may be possible that the true duplicates of some original records are filtered away.

#### *D. Other Applications*

Apart from the scenarios described above, OSA distance can be applied in many other scenarios where triangular inequality is a must (e.g. metric spaces [12] or k-nearest neighbors algorithms [14]).

Also, due to its low practical cost, the OSA distance is also suitable for scenarios where a large number of string distance calculations have to be done (e.g. clustering algorithms [33] or sequential pattern mining methods [34]).

As illustrative examples of the large number of potential applications of the OSA distance, we describe here two alternative scenarios.

- *Gene sequential pattern mining.* Such data mining applications extract the genetic information from a set of DNA chips implanted in different individuals (usually, monkeys) to be studied. A common problem with these chips is that the intensity of the extracted participant genes differs from one individual to the other, in the studied biological processes. When researchers working with this data want to group biological processes with similar interaction of genes, they have to cluster sequences of symbols sharing a large number of symbols (genes) but with different intensity order. This problem could be addressed by applying a clustering algorithm which uses the OSA distance as the clustering criterion.
- *Metric spaces data structures.* Another interesting application of OSA distance is in the creation, initialization and maintenance of metric spaces data structures. Such structures need to compute a large number of distances between the elements inside the metric space and a set of vantage points. These structures are mainly used for solving k-nearest neighbors queries in an efficient way, by using the distances to some vantage points and the triangular inequality for discarding far points that cannot be a closer neighbor.

## VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a new comparison function (the OSA distance) for sequences of symbols. We have proved that our function is a real distance, showing that it satisfies the triangular inequality property. We have proposed two different algorithms to compute the OSA distance of two sequences. We have also described a set of experiments for long DNA sequences and for record linkage across synthetic databases, showing that the OSA distance is faster than other well-known and widely used string comparison functions. From our experiments we can conclude that the OSA distance always outperforms in terms of recall and execution time the Jaro-Winkler distance, maybe the fastest (non-trivial) string comparison function considered up to now. We have also showed that the quality values achieved by the new distance are comparable to the ones obtained by other existing protocols, based on the Edit distance.

In our experiments, no advantage is taken from the fact that the computed comparison function is a mathematical distance. As future work, we will try to find real scenarios where the total number of comparisons to be done can be decreased when the comparison function satisfies the triangular inequality. In such situations, the difference between the execution times achieved by using the OSA distance or other ‘distances’ (like Jaro-Winkler) should be even bigger.

## REFERENCES

- [1] F. Hoerndli, D. C. David, and J. Götz, “Functional genomics meets neurodegenerative disorders : : Part ii: Application and data integration,” *Progress in Neurobiology*, vol. 76, no. 3, pp. 169–188, 2005.
- [2] N. Shoval, G. K. Auslander, T. Freytag, R. Landau, F. Oswald, U. Seidl, H.-W. Wahl, S. Werner, and J. Heinik, “The use of advanced tracking technologies for the analysis of mobility in alzheimer’s disease and related cognitive diseases,” *BMC Geriatrics*, vol. 8:7, 2008.
- [3] R. Agrawal and R. Srikant, “Mining sequential patterns,” in *Proceedings of the Eleventh International Conference on Data Engineering*, 1995, pp. 3–14.
- [4] G. Dong and J. Pei, *Sequence Data Mining*. Springer, 2007.
- [5] C. Gómez-Alonso and A. Valls, “A similarity measure for sequences of categorical data based on the ordering of common elements,” in *Proc. of the Int. Conf. of Modeling Decisions for Artificial Intelligence (MDAI)*, ser. Lecture Notes on Artificial Intelligence. Springer, 2008, pp. 134–145.
- [6] R. W. Hamming, “Error detecting and error correcting codes,” *Bell System Technical Journal*, vol. 26, no. 2, pp. 147–160, 1950.
- [7] M. Jaro, “Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida,” *Journal of the American Statistical Association*, vol. 84, pp. 414–420, 1989.
- [8] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Soviet Physics Doklady*, vol. 10, no. 707–710, 1966.
- [9] G. Navarro, “A guided tour to approximate string matching,” *ACM Computing Surveys*, vol. 33, no. 1, pp. 31–88, 2001.
- [10] H. Saneifar, S. Bringay, A. Laurent, and M. Teisseire, “S2mp: Similarity measure for sequential patterns,” in *Proc. 7th Australasian Data Mining Conference (AusDM)*, 2008, pp. 95–104.
- [11] P. Selliers, “The theory and computation of evolutionary distances: pattern recognition,” *Journal of Algorithms*, pp. 359–373, 1980.
- [12] E. Chávez, G. Navarro, R. Baeza-yates, and J. L. Marroquín, “Searching in metric spaces,” *ACM Computing Surveys*, vol. 33, pp. 273–321, 1999.
- [13] A. K. Jain, M. N. Murty, and P. Flynn, “Data clustering: a review,” *ACM Computing Surveys*, vol. 31(3), pp. 264–323, 1999.
- [14] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [15] F. J. Damerau, “A technique for computer detection and correction of spelling errors,” *Communications of the ACM*, vol. 7, no. 3, pp. 171 – 176, 1964.
- [16] E. Ristad and P. Yianilos, “Learning string edit distance,” *IEEE Transactions on Pattern Recognition and Machine Intelligence*, vol. 20, no. 5, pp. 522–532, 1998.
- [17] R. A. Wagner and M. J. Fischer, “The string-to-string correction problem,” *Journal of ACM*, vol. 21, no. 1, pp. 168–173, 1974.
- [18] E. Ukkonen, “On approximate string matching,” in *Proc. of the Int. FCT-Conference on Fundamentals of Computation Theory*, 1983, pp. 487–495.
- [19] H. Berghel and D. Roach, “An extension of Ukkonen’s enhanced dynamic programming asm algorithm,” *ACM Transactions Information Systems*, vol. 14, no. 1, pp. 94–106, 1996.
- [20] NCBI BLAST databases, “<ftp://ftp.ncbi.nih.gov/blast/db/FASTA/>.”

- [21] Catalan Official Statistics Institute (IDESCAT), “<http://www.idescat.cat/en/>.”
- [22] K. Kukich, “Techniques for automatically correcting words in text,” *ACM Computing Surveys*, vol. 24, no. 4, pp. 377–439, 1992.
- [23] W. E. Winkler, “Data cleaning methods,” in *Proc. of the ACM Workshop on Data Cleaning, Record Linkage and Object Identification*, 2003.
- [24] V. Torra and J. Domingo-Ferrer, “Record linkage methods for multidatabase data mining,” in *Information Fusion in Data Mining*. Springer, 2003, pp. 101–132.
- [25] strcmp.c Original C Implementation of Jaro-Winkler distance, “<http://www.census.gov/geo/msb/stand/strcmp.c>.”
- [26] N. Koudas, S. Sarawagi, and D. Srivastava, “Record linkage: similarity measures and algorithms,” in *Proc. of the ACM Int. Conf. on Management of data (SIGMOD)*, 2006, pp. 802–803.
- [27] W. Winkler, “The state of record linkage and current research problems,” Statistical Research Division, U.S. Bureau of the Census, Tech. Rep., 1999.
- [28] M. Hernandez and S. Stolf, “Real-world data is dirty: data cleansing and the merge/purge problem,” *Journal of Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 9–37, 1998.
- [29] L. Gu, R. Baxter, D. Vickers, and C. Rainsford, “Record linkage: Current practice and future directions,” CSIRO Mathematical and Information Sciences, Tech. Rep. 03/83, 2003.
- [30] L. Jin, C. Li, and S. Mehrotra, “Efficient record linkage in large data sets,” in *Proc. Int. Conf. on Database Systems for Advanced Applications (DASFAA)*, 2003, pp. 137–146.
- [31] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, “Robust and efficient fuzzy match for online data cleaning,” in *Proc. of the ACM SIGMOD Int. Conf. on Management of data (SIGMOD)*, 2003, pp. 313–324.
- [32] C. Xiao, W. Wang, and X. Lin, “Ed-join: An efficient algorithm for similarity join with edit distance constraints,” in *Proc. of the Very Large DataBase (VLDB)*, 2008, pp. 933–944.
- [33] A. Jain, M. Murty, and P. Flynn, “Data clustering: a review,” *ACM Computing Surveys*, vol. 31, no. 3, pp. 264 – 323, 1999.
- [34] R. Agrawal and R. Srikant, “Mining sequential patterns,” in *Int. Conf. Data Engineering (ICDE)*, 1995, pp. 3–14.

**Javier Herranz** obtained his Ph.D. in Applied Mathematics in 2005, in the Technical University of Catalonia (UPC, Barcelona, Spain). After that, he spent 9 months in the École Polytechnique (France), 9 months in the Centrum voor Wiskunde en Informatica (CWI, The Netherlands) and 2 years in IIIA-CSIC (Bellaterra, Spain), as a post-doctoral researcher. Currently he enjoys a Ramón y Cajal grant by the Spanish Ministry of Education and Sciences, for post-doctoral research in the group MAK (Dept. Applied Mathematics IV, UPC). His research interests are related to cryptography and privacy of databases.

**Jordi Nin** holds a Ph.D. (2008) in Computer Science by the Autonomous University of Barcelona (UAB). He works as a post-doctoral researcher at the Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS) in Toulouse, France. His fields of interests are privacy technologies, machine learning and soft computing tools. He has been involved in several research projects funded by the Catalan and Spanish governments and the European Community. His research has been published in specialized journals and major conferences (around 50 papers).

**Marc Solé** holds a Ph.D. (2009) in Computer Science by the Technical University of Catalonia (UPC), where he is also an Assistant Professor in the Computer Science Faculty of Barcelona. His Ph.D. thesis was focused on automated formal verification of timed concurrent systems. His research interests include formal verification, process mining, data anonymization and UAS. He belongs to the ICARUS research group.